042390.P9774 PATENT

UNITED STATES PATENT APPLICATION

for

METHOD AND APPARATUS FOR CONSTRUCTING HOST PROCESSOR SOFT DEVICES INDEPENDENT OF THE HOST PROCESSOR OPERATING SYSTEM

Applicants:

Eric Cota-Robles
Stephen Chou
Stalinselvaraj Jeyasingh
Alain Kagi
Michael Kozuch
Richard Uhlig

prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN
12400 Wilshire Boulevard
Los Angeles, CA 90026-1026
(408) 720-8598

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number EL672750725US
Date of Deposit March 30, 2001
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231.

Michelle Begay
(Typed or printed name of person mailing paper or fee)

Michelle Begay
(Signature of person mailing paper or fee)

METHOD AND APPARATUS FOR CONSTRUCTING HOST PROCESSOR SOFT DEVICES INDEPENDENT OF THE HOST PROCESSOR OPERATING SYSTEM

Field of the Invention

The present invention relates generally to virtual machines, and more specifically to constructing host processor soft devices independent of the host processor operating system.

Background of the Invention

A host-based soft device is a hardware device that includes a host software component and a reduced functionality hardware component: respectively a host processor device driver and a "residual" fixed function device. The host processor device driver replaces portions of a fixed function device in a personal computing system (e.g., personal computers, personal digital assistants, etc.). Conventional fixed function devices incorporate one or more dedicated general and special purpose processors (e.g., a microcontroller, a digital signal processor, etc.) and a limited amount of dedicated random access memory (RAM) and read-only memory (ROM). In a host-based soft device, software that executes on one or more dedicated processors is replaced by a host processor device driver that uses main system memory. Typically, a host-based soft device is less expensive than the conventional fixed function devices because the "residual" fixed function device that requires fewer dedicated processors and smaller amounts of

dedicated RAM and ROM. In addition, because the host processor is far more powerful than the dedicated processors and has access to more memory, a host-based soft device is often able to use more powerful algorithms, thereby improving performance, increasing efficiency and/or providing higher benchmark scores.

Despite the above advantages of host-based soft devices, their implementation has been limited almost entirely to Microsoft Windows operating systems due to the large market share of Windows family of operating systems, leaving other personal computer operating systems (e.g., BeOS) without available host-based soft devices. In addition, the development of a soft device for Windows has not been an easy process. Recent experience with Windows-based soft devices has shown the inability of Microsoft to develop a stable device driver model. For instance, during the year of 1999, soft modem vendors were simultaneously required to port device software to four different soft modem device drivers for each soft device: specifically, VxD for Windows 98, WDM 1.0 for Windows 98SE, NT for Windows NT 4.0 and WDM 2.0 for Beta releases of Windows 2000. In addition, Windows-based soft devices suffer from excessive latency in the host operating system because Windows operating systems lack real-time quality of service (QoS) guarantees.

Therefore, it would be advantageous to construct a host processor soft device that would be independent of the host processor operating system.

Brief Description of the Drawings

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

Figure 1 illustrates one embodiment of a virtual-machine environment;

Figure 2 is a block diagram of a system in which a soft device driver is implemented in a virtual machine monitor (VMM), according to one embodiment of the present invention;

Figure 3 is a flow diagram of a method for constructing a soft device via a VMM implementation, according to one embodiment of the present invention;

Figure 4 is a block diagram of a system in which a software component of a soft device is implemented in a virtual machine, according to one embodiment of the present invention;

Figure 5 is a flow diagram of a method for constructing a soft device via a virtual machine implementation, according to one embodiment of the present invention;

Figures 6A – 6C illustrate various embodiments of system 400 of **Figure 4**; and

Figure 7 is a block diagram of one embodiment of a processing system.

Description of Embodiments

Methods and apparatus for constructing host processor soft devices independent of the host processor operating system are described. In the following descriptions, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention can be practiced without these specific details.

Some portions of the detailed descriptions that follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussions, it is appreciated that

throughout the present invention, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, may refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

The present invention also relates to apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, and magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), EPROMs, EEPROMs, magnetic or optical cards, or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus. Instructions are executable using one or more processing devices (e.g., processors, central processing units, etc.).

The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general purpose machines may be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized

apparatus to perform the required method steps. The required structure for a variety of these machines will appear from the description below. In addition, the present invention is not described with reference to any particular programming language. It will be appreciated that a variety of programming languages may be used to implement the teachings of the invention as described herein.

In the following detailed description of the embodiments, reference is made to the accompanying drawings that show, by way of illustration, specific embodiments in which the invention may be practiced. In the drawings, like numerals describe substantially similar components throughout the several views. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention. Other embodiments may be utilized and structural, logical, and electrical changes may be made without departing from the scope of the present invention. Moreover, it is to be understood that the various embodiments of the invention, although different, are not necessarily mutually exclusive. For example, a particular feature, structure, or characteristic described in one embodiment may be included within other embodiments. The following detailed description is, therefore, not to be taken in a limiting sense, and the scope of the present invention is defined only by the appended claims, along with the full scope of equivalents to which such claims are entitled.

Figure 1 illustrates one embodiment of a virtual-machine environment 100, in which the present invention may operate. The host platform 116 is capable of executing a virtual machine monitor (VMM) 112. The VMM 112,

though typically implemented in software, exports a bare machine interface to higher level software. The interface is exported as one or more virtual machines (VMs) and may mirror the actual host hardware platform, so that it is virtualized. Alternatively, the interface exported by the VMM 112 may differ in some or all respects so that a different platform is emulated. The higher level software may comprise a standard or real-time OS, although the invention is not limited in scope in this respect and, alternatively, for example, the VMM 104 may be run within, or on top of, another VMM. VMMs and their typical features and functionality are well-known by those skilled in the art and may be implemented, for example, in software, firmware or by a combination of various techniques.

As described above, the VMM 112 presents to other software (i.e., "guest" software) the abstraction of one or more virtual machines (VMs). **Figure 1** shows VMs 102 and 114. VMs 102 and 114 run their own guest operating systems: guest OSes 104 and 106. The guest OS is provided with the illusion of executing on the host platform, rather than in a virtual platform. In one embodiment, the virtual abstraction presented to the guest OS matches the characteristics of the host platform 116. Alternatively, the virtual abstraction presented to the guest OS differs from the characteristics of the host platform 116. The VMM 112 provides protection between VMs 102 and 114 and can observe and restrict the activities of the VMs 102 and 114.

The present invention provides a mechanism for constructing a host processor soft device independent of the host processor OS (host OS) in a virtual-machine environment. **Figure 2** illustrates one embodiment of a

system, in which a soft device is constructed in a manner that does not depend on any particular host OS.

Referring to **Figure 2**, a software component 208 of the soft device is implemented as a soft device driver directly in the VMM 122. The soft device driver 208 controls a hardware component of the soft device identified as a "residual" fixed function device 204. Accordingly, any guest OS 106 or 114 (e.g., Windows or Linux) can access the soft device driver 208 and use the soft device as needed.

The soft device driver 208 can be implemented in the same manner as a host-based soft device driver would be implemented in a conventional host OS. However, while a host OS soft device driver operates in such a way as to export a standard device abstraction to the host OS, the soft device driver 208 implemented in the VMM 112 exports an emulation of a fixed function hardware device to VMs 102 and/or 114. The guest OS 106 or 114 then runs a standard device driver and attempts to read from, and write to, hardware registers of an emulated fixed function hardware device. The soft device driver 208 recognizes these attempts of the guest OS 106 or 114 to interact with hardware registers that may not exist and emulates the functionality of a fixed function hardware device.

In some embodiments (e.g., when the corresponding functionality is performed by the host processor), the soft device driver 208 emulates the functionality of a fixed function device by performing the requested computations without any notification to the residual fixed function device 204.

In other embodiments, the soft device driver may pass the read or write command without alteration to the residual fixed function device 204 to perform an operation requested by the guest OS 106 or 114.

In still other embodiments, the soft device driver 208 may perform some of the requested computations and may then perform one or a whole series of similar or completely different operations on the actual hardware registers of the residual fixed function device 204 in order to complete the task that the guest OS 106 or 114 requested.

In yet other embodiments, the residual fixed function device 204 is capable of directly accessing system memory. In one embodiment, the soft device driver 208 performs zero or more transformations on data received from guest OS 106 or 114 before directly transferring the data to system memory at a predefined location from which the residual fixed function device 204 will fetch the data in order to effect the operation requested by the guest OS 106 or 114. In another embodiment, the soft device driver 208 performs zero or more transformations on data directly transferred to system memory at a predefined location by the residual fixed function device 204, said transformed data then being transferred to the guest OS 106 or 114 in order to complete the requested operation.

Figure 3 is a flow diagram of a method 300 for constructing a soft device via a VMM implementation, according to one embodiment of the present invention. Method 300 begins with implementing a soft device driver in a VMM (processing block 304). As described above, it may be done in the same well known in the art manner as for implementing a soft device driver

in a conventional host OS.

Subsequently, when a request to use the soft device is received from a virtual machine coupled to the VMM (processing block 306), the soft device is made available for use by this virtual machine (processing block 308). In one embodiment, making the soft device available for use by the virtual machine includes exporting an emulation of a fixed function hardware device to this virtual machine, recognizing the attempts of the virtual machine to read and write to hardware registers of the emulated device, and emulating the functionality of a fixed function hardware device to perform the read and or write operation requested by the virtual machine. In one embodiment, this functionality is emulated by the soft device driver performing the requested computations without any notification to a hardware component of the soft device (i.e., a residual fixed function device). In an alternative embodiment, the functionality is emulated by the hardware component of the soft device which receives the read or write command from the soft device driver and executes this command to perform an operation requested by the virtual machine. In still another embodiment, the soft device driver performs some of the requested computations and then performs one or a whole series of similar or completely different operations on the actual hardware registers of the residual fixed function device in order to complete the task intended by the virtual machine. In yet another embodiment, the residual fixed function device 204 is capable of directly accessing system memory and transfers data to or from system memory at predefined locations, and the soft device driver manipulates data stored in memory according to the operation requested by

the virtual machine. For instance, the soft device driver performs the write command received from the virtual machine by transforming the data in the memory appropriately and then transferring the requested data to the memory using the direct memory access (DMA) technique.

It should be noted that any combination of the above actions can be used by the soft device driver to emulate the functionality of a fixed function hardware device without loss of generality. In one embodiment, the virtual machine runs an arbitrary OS (e.g., Linux), for which no soft device drivers exist on the market. Accordingly, the VMM implementation of a soft device driver enables a soft device to be accessed from a wide variety of host OSes, including host OSes for which no appropriate soft device drivers exist on the market. In addition, such soft device drivers do not suffer from latency problems in the host OS because they are implemented beneath the host OS.

Figure 4 is a block diagram of a system in which a software component of a soft device is implemented in a virtual machine, according to one embodiment of the present invention. Referring to **Figure 4**, a software component 418 of a soft device is implemented in a dedicated virtual machine 408. The dedicated virtual machine 408 runs a guest OS 416, and a separate virtual machine 406 runs a guest OS 410 with which the user interacts. The dedicated virtual machine 408 is scheduled for execution on the host processor using any of the methods known in the art that are sufficient to provide the virtual machine 408 with real-time quality of service (QoS) guarantees.

The VMM 112 ensures that the soft device is available for use by the

virtual machine 406 when needed. In particular, the VMM 112 provides such communication between the virtual machines 406 and 408 as to make the soft device controlled by the dedicated virtual machine 408 available to the virtual machine 406. This communication is accomplished using a virtualized connectivity means 404 that represents a software version of such connectivity means as a local area network (LAN) (presented to each of VMs 406 and 408 as a virtual network interface card (NIC)), a serial communications link (presented to each of VMs 406 and 408 as a virtual serial port (e.g., COM 1)), a universal serial bus (USB) to USB bridge (presented to each of VMs 406 and 408 as a virtual USB "host controller" connected to a USB to USB bridge device), a peripheral component interconnect (PCI) bus, a USB connection (presented to VM 406 as a virtual USB "host controller" and to VM 408 as a virtual USB bus interface), etc. The virtual machine implementation of a soft device allows construction of a soft device that is not only OS independent but also VMM independent. That is, the soft device driver does not need to be written for use with a particular VMM. Again, such a soft device is immune to latency in the host OS because its driver is executed in a dedicated virtual machine.

In one embodiment, the dedicated virtual machine 408 includes reflection software 414 (e.g., a reflection application or driver) which links the software component 418 (e.g., a soft device driver) and the virtualized connectivity means together. For instance, the reflection software 414 may be configured to read from the serial port COM1 and write to the soft device driver, and vice versa. In alternative embodiments, a direct communication

between the software component 418 and the virtualized connectivity means is provided, thereby obviating the need for the reflection software 414.

In one embodiment, the dedicated virtual machine 408 is configured to match the underlying host platform. In this embodiment, the dedicated virtual machine 408 may execute a conventional soft device driver that is written for the OS 416. For instance, the dedicated virtual machine may execute a soft modem driver under Windows NT 4.0. The virtual machine 406 may instead run an arbitrary OS for which no soft device driver need be or have been written. For instance, the virtual machine 406 may run Linux, Unix, BeOS, or any other OS. When the dedicated virtual machine 408 is configured to match the underlying host platform, the VMM 112 may present the dedicated virtual machine 408 to the virtual machine 406 as an external conventional fixed function device using a virtualized serial communications port (COM1) or virtualized USB "host controller" or as an internal conventional fixed function device using a virtualized local bus (e.g., a PCI bus). These embodiments of the present invention will be described in greater detail below in conjunction with **Figure 6A**. Alternatively, the VMM 112 emulates a network communication between the dedicated virtual machine 408 and other one or more virtual machines coupled to the VMM 112 using a virtualized LAN, as will be described in greater detail below in conjunction with **Figure 6B**.

In an alternative embodiment, the dedicated virtual machine 408 is configured to match an existing hardware device (e.g., a PCI card, a USB device, etc.), and may differ significantly from the host platform. In this

embodiment which is described in more detail below in conjunction with **Figure 6C**, the VMM 112 emulates communication between the virtual machines 406 and 408 using a virtualized PCI bus or a virtualized USB connection.

Figure 5 is a flow diagram of a method 500 for constructing a soft device via a virtual machine implementation, according to one embodiment of the present invention. Method 500 begins at processing block 504 with implementing a software component of a soft device in a dedicated VM which is coupled to a virtual machine monitor. In one embodiment, the software component is a soft device driver. Alternatively, the software component is an existing software component that has been previously implemented for a bare hardware platform. As described above, the user interacts with another virtual machine (referred to as a main virtual machine) that is also coupled to the VMM. In one embodiment, the dedicated virtual machine is configured to match the underlying host platform. Alternatively, the dedicated virtual machine is configured to match an existing hardware device and does not match the host platform. In one embodiment, the main virtual machine runs an arbitrary OS for which no soft device drivers exist on the market.

Subsequently, when a request to use the soft device is received from the main virtual machine (processing block 506), the soft device is made available for use by the main virtual machine (processing block 508).

In one embodiment, making the soft device available for use by the main virtual machine includes presenting the dedicated virtual machine to the main virtual machine as either an external or an internal conventional

fixed function device and emulating communication between the dedicated virtual machine and the main virtual machine. This communication is emulated by providing a virtualized connectivity means, linking the virtualized connectivity means to the soft device driver, trapping all accesses by one of the two virtual machines to the virtualized connectivity means, and re-directing these accesses to the other of the two virtual machines. In one embodiment, the virtualized connectivity means is a virtualized serial communications link. Alternatively, the virtualized connectivity means is a virtualized USB to USB bridge or a virtualized local bus such as a PCI bus.

In another embodiment, making the soft device available for use by the main virtual machine includes emulating a virtual network that connects the dedicated virtual machine to the main virtual machine and to any other virtual machine coupled to the VMM using a virtualized LAN. The communication between the virtual machines is then emulated by trapping all accesses to the virtualized LAN and reflecting them to the appropriate virtual machine.

In yet another embodiment, making the soft device available for use by the main virtual machine includes configuring the dedicated virtual machine to match an existing hardware device, such as a PCI card, a USB device or any other standard PC peripheral device, and emulating communication between the dedicated virtual machine and the main virtual machine using, respectively, a virtualized PCI bus, a virtualized USB connection, or a virtualized bus, connection or link appropriate to the device.

The above three embodiments will now be described in more detail in

conjunction with **Figures 6A – 6C**. Referring to **Figure 6A**, a dedicated virtual machine 608 is configured to match the underlying host platform. The dedicated virtual machine 608 executes a conventional soft device driver 618 that is written for the OS 616 (e.g., Windows NT 4.0). The main virtual machine 606 runs an arbitrary OS 610 for which no soft device drivers need exist on the market.

In one embodiment, the VMM 112 presents the dedicated virtual machine 608 to the main virtual machine 606 as an external fixed function hardware device (e.g., a "hard" or standard modem) by providing a virtualized serial port 404 (e.g., COM 1), trapping all accesses by the main virtual machine 606 to this virtualized serial port and re-directing them to the dedicated virtual machine 608. Similarly, the VMM 112 may trap the proper set of I/O writes by the dedicated virtual machine 608 and redirect these writes to the main virtual machine 606 as to present them as data from the serial port. In one embodiment, the dedicated virtual machine 608 executes reflection software (e.g., an application or device driver) 614 to link the soft device driver 618 and the virtualized serial port 404 together. For instance, the reflection software 614 may read from the virtualized serial port 404 and write to the soft device driver 618 and vice versa. In one embodiment, the dedicated virtual machine 608 is assigned an Interrupt Request (IRQ) for the main virtual machine 606. The VMM 112 then traps a predefined software instruction or a trap, fault, exception or software interrupt that occurs under predefined conditions in the dedicated virtual machine 608 and reflects each such instruction or occurrence of a trap fault, exception or software interrupt

when the predefined conditions hold as an interrupt of the appropriate IRQ to the virtualized Programmable Interrupt Controller (PIC) of the main virtual machine 606.

In another embodiment, the VMM 112 presents the dedicated virtual machine 608 to the main virtual machine 606 as an internal fixed function hardware device by providing a virtualized USB to USB bridge 404, presenting the virtualized USB to USB bridge 404 to each virtual machine 606 and 608 as a USB "host controller" connected to a USB to USB bridge device, trapping all accesses by the main virtual machine 606 to this virtualized USB host controller 404 and re-directing them to the dedicated virtual machine 608. In addition, the VMM 112 traps accesses by the dedicated virtual machine 608 to the USB host controller 404 and redirects these accesses to the main virtual machine 606 as to present them as data from the USB host controller.

In yet another embodiment, the VMM 112 presents the dedicated virtual machine 608 to the main virtual machine 606 as an internal fixed function hardware device or a PCI device by providing a virtualized local bus 404 (e.g., a PCI bus), and trapping and redirecting all accesses to this virtualized local bus 404 in the manner described above.

Referring to **Figure 6B**, a dedicated virtual machine 638 is configured to match the underlying host platform. The dedicated virtual machine 638 executes a conventional soft device driver 648 that is written for the OS 640 (e.g., Windows NT 4.0). The main virtual machine 636 runs an arbitrary OS 630 for which no soft device drivers exist on the market. Other virtual

machines (e.g., a virtual machine 646) also run arbitrary OSes. In this embodiment, the VMM 112 does not present the dedicated virtual machine to other virtual machines as a fixed function hardware device but instead presents it as a PC connected to the other virtual machines via a virtual communication network 642 (e.g., a virtualized LAN presented to each virtual machine 636, 638 or 646 by the VMM 112 as a corresponding virtual NIC). The virtual machines can then communicate to each other via the virtual network controlled by the VMM 112. For instance, the main virtual machine 636 may send a request to the dedicated virtual machine 638 to set a service to use the soft device via the virtual network. In one embodiment, reflection software 634 is used to link the soft device driver 648 and a corresponding virtual NIC together. Alternatively, conventional bridge and routing software can be used for this purpose, thereby obviating the need for the reflection software.

Referring to **Figure 6C**, a dedicated virtual machine 658 is configured to match an existing hardware device (e.g., a PCI card). As such, the dedicated virtual machine 658 differs from the host platform. For instance, the dedicated virtual machine 658 may only have a processor, RAM, ROM and a connection to a platform bus (e.g., PCI) or to an external bus (e.g., USB). The dedicated virtual machine 658 may model both RAM and ROM using system RAM assigned to it by the VMM 112.

The VMM 112 emulates communication between the two virtual machines by trapping I/O accesses by the dedicated virtual machine 658 to the virtualized PCI bus or virtualized USB connection 654 and redirecting

them to the main virtual machine 655, and vice versa.

As illustrated in **Figure 6C**, the dedicated virtual machine 658 runs RTOS 666 which can be chosen to match the OS run by one or more of the processors on a conventional fixed function hardware device. The functionality performed by the software that executes on these processors can be executed in the virtual machine 658 and the residual fixed function hardware 652 can be substituted for the conventional fixed function hardware device. Residual fixed function hardware 652 could be a PCI card, a USB device, or any other standard PC peripheral device.

As opposed to other OSes (e.g., Windows OSes), RTOS 666 does not require a change to a soft device driver each time the PC OS such as Windows is modified. Instead, the dedicated virtual machine 658 that closely matches the PCI card, USB or other PC peripheral device can execute some or all of the software of the original fixed function hardware device. With availability of on chip co-processors (e.g., a digital signal processor (DSP)), the virtual machine may make direct use of on chip co-processors to provide an extended Instruction Set Architecture (ISA) that closely matches the ISA of the processor(s) on the original PCI card. As result, the soft device software that executes on the host processor need not to be different from the software which executes on the original fixed function device, thereby simplifying porting and speed of development of soft devices.

The present invention provides the software vendors with an option of constructing soft devices by reconfiguring a dedicated virtual machine, rather than rewriting soft device driver software. For instance, both approaches can

be analyzed and the most efficient approach can be chosen based on the costs involved. For complex conventional fixed function hardware devices multiple virtual machines can be configured, with each VM running the RTOS used by one or more of the processors of the conventional fixed function hardware device and the VMM emulating communication between the processors on the conventional fixed function hardware device using any of a variety of well known techniques including the communication techniques described above. For example, a complex fixed function hardware device may have a serial link between two of its processors.

Figure 7 is a block diagram of one embodiment of a processing system. Processing system 700 includes processor 720 and memory 730. Processor 720 can be any type of processor capable of executing software, such as a microprocessor, digital signal processor, microcontroller, or the like. Processing system 700 can be a personal computer (PC), mainframe, handheld device, portable computer, set-top box, or any other system that includes software.

Memory 730 can be a hard disk, a floppy disk, random access memory (RAM), read only memory (ROM), flash memory, or any other type of machine medium readable by processor 720. Memory 730 can store instructions for performing the execution of the various method embodiments of the present invention such as methods 300 and 500 (**Figures 3 and 5**).

It is to be understood that the above description is intended to be illustrative, and not restrictive. Many other embodiments will be apparent to those of skill in the art upon reading and understanding the above

description. The scope of the invention should, therefore, be determined with reference to the appended claims, along with the full scope of equivalents to which such claims are entitled.